diia.gov.ua

Diia: Mobile Application Bug Bounty Program
DIIA

Diia: Mobile Application Bug Bounty Program

Bugcrowd On-Demand program results

Report created on December 17, 2020

bugcrowd

Prepared by

Michael Perry, Security Solutions Consultant
michael.perry@bugcrowd.com

Table of contents

1 Executive summary **3**

2 Reporting and methodology **4**

Background 4

3 Targets and scope **5**

Scope 5

4 Findings summary **6**

Findings by severity 6

Risk and priority key 7

Findings table 8

5 Vulnerability details **9**

6 Appendix **16**

Submissions over time 16

Submissions signal 16

Bug types overview 18

7 Closing statement **19**

Executive summary

Diia: Mobile Application Bug Bounty Program engaged Bugcrowd, Inc. to perform an On-Demand Bounty Program, commonly known as a crowd-sourced penetration test.

An On-Demand Bounty Program is a cutting-edge approach to an application assessment or penetration test. Traditional penetration tests use only one or two personnel to test an entire scope of work, while an On-Demand Bounty leverages a crowd of security researchers. This increases the probability of discovering esoteric issues that automated testing cannot find and that traditional vulnerability assessments may miss in the same testing period.

The purpose of this program was to identify security vulnerabilities in the targets listed in the targets and scope section. Once identified, each vulnerability was rated for technical impact defined in the findings summary section of the report.

Testing for **Diia: Mobile Application Bug Bounty Program's** targets occurred during the period of: **12/08/2020 – 12/15/2020**.

For this On-Demand Program, **83** researchers were invited to participate; **27** accepted the invitation. Submissions were received from **4** unique researchers.

The continuation of this document summarizes the findings, analysis, and recommendations from the On-Demand Bounty Program performed by Bugcrowd for **Diia: Mobile Application Bug Bounty Program**.

This report is just a summary of the information available.

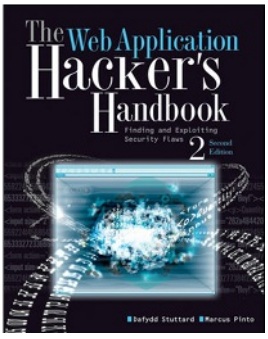
All details of the program's findings — comments, code, and any researcher provided remediation information — can be found in the Bugcrowd Crowdcontrol platform.

Background

The strength of crowdsourced testing lies in multiple researchers, the pay-for-results model, and the varied methodologies that the researchers implement. To this end, researchers are encouraged to use their own individual methodologies on Bugcrowd On-Demand programs.

The workflow of every penetration test can be divided into the following four phases:

Bugcrowd researchers who perform web application testing and vulnerability assessment usually subscribe to a variety of methodologies following the highlighted workflow, including the following:



Scope

Prior to the On-Demand program launching, Bugcrowd worked with Diia: Mobile Application Bug Bounty Program to define the Rules of Engagement, commonly known as the program brief, which includes the scope of work. The following targets were considered explicitly in scope for testing:

```
DIIA Android Application
```

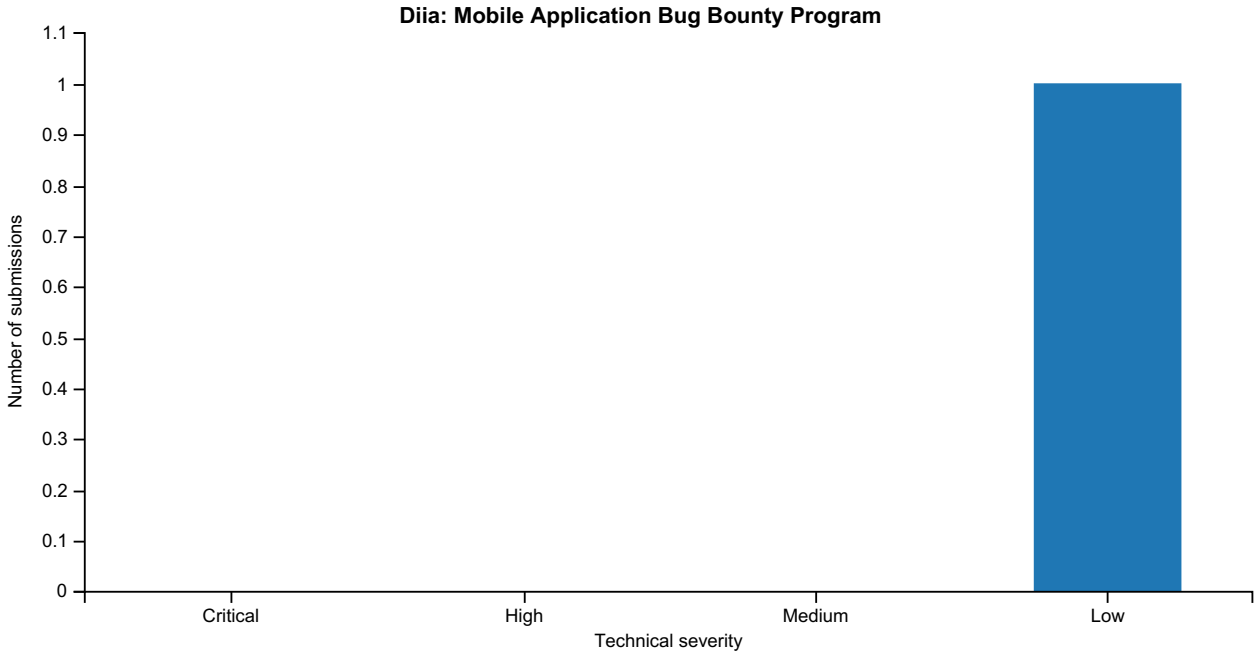
```
DIIA iOS Application
```

```
DIIA API
```

All details of the program scope and full program brief can be reviewed in the Program Brief.

Findings by severity

The following chart shows all valid assessment findings from the program by technical severity.



Risk and priority key

The following key is used to explain how Bugcrowd rates valid vulnerability submissions and their technical severity. As a trusted advisor Bugcrowd also provides common "next steps" for program owners per severity category.

TECHNICAL SEVERITY	EXAMPLE VULNERABILITY TYPES
Critical Critical severity submissions (also known as "P1" or "Priority 1") are submissions that are escalated to **Diia: Mobile Application Bug Bounty Program** as soon as they are validated. These issues warrant the highest security consideration and should be addressed immediately. Commonly, submissions marked as Critical can cause financial theft, unavailability of services, large-scale account compromise, etc.	• Remote Code Execution • Vertical Authentication Bypass • XML External Entities Injection • SQL Injection • Insecure Direct Object Reference for a critical function
High High severity submissions (also known as "P2" or "Priority 2") are vulnerability submissions that should be slated for fix in the very near future. These issues still warrant prudent consideration but are often not availability or "breach level" submissions. Commonly, submissions marked as High can cause account compromise (with user interaction), sensitive information leakage, etc.	• Lateral authentication bypass • Stored Cross-Site Scripting • Cross-Site Request Forgery for a critical function • Insecure Direct Object Reference for an important function • Internal Server-Side Request Forgery
Medium Medium severity submissions (also known as "P3" or "Priority 3") are vulnerability submissions that should be slated for fix in the major release cycle. These vulnerabilities can commonly impact single users but require user interaction to trigger or only disclose moderately sensitive information.	• Reflected Cross-Site Scripting with limited impact • Cross-Site Request Forgery for an important function • Insecure Direct Object Reference for an unimportant function
Low Low severity submissions (also known as "P4" or "Priority 4") are vulnerability submissions that should be considered for fix within the next six months. These vulnerabilities represent the least danger to confidentiality, integrity, and availability.	• Cross-Site Scripting with limited impact • Cross-Site Request Forgery for an unimportant function • External Server-Side Request Forgery
Informational Informational submissions (also known as "P5" or "Priority 5") are vulnerability submissions that are valid but out-of-scope or are "won't fix" issues, such as best practices.	• Lack of code obfuscation • Autocomplete enabled • Non-exploitable SSL issues

Bugcrowd's Vulnerability Rating Taxonomy

More detailed information regarding our vulnerability classification can be found at: https://bugcrowd.com/vrt

Findings table

The following table lists all valid assessment findings from the program:

Title	VRT	Duplicates	Priority	State	Link
IDOR via license plate at endpoint /api/v1/documents/vehicle-insurance	Broken Access Control (BAC)	-	P4	TRIAGED	
App Crash on Malformed QR Code Read	Application-Level Denial-of-Service (DoS)	-	P5	TRIAGED	

Vulnerability details

bugcrowd

This section outlines the full submission data for each valid finding. These findings are unaltered from their original state from the researcher. Due to the competitive nature and gamification of crowd-sourced security assessments, some typos or grammar errors may occur. Each finding is headlined with the submission title and priority followed by more detailed vulnerability information based on the type of finding submitted. Several other fields may appear based on the context and VRT classification selected by a researcher.

Such details may include the following:

Description:
This section appears above the "Reference Number" as a free form area for the researcher to describe the context of the submission.

Reference number:
Submission unique Identifier visible to researchers.

VRT:
The Vulnerability Rating Taxonomy is the baseline guide used for classifying technical severity.

Bug URL:
This is the full URL/URI of where the vulnerability took place.

Extra info:
A free form area for the researcher to add additional information to the submission.

HTTP request:
This is a text block with the full HTTP(S) request that triggered the vulnerability, including all its associated headers and cookie information.

Additional details:
Several other fields may appear based on the context and VRT classification selected by a researcher. Bugcrowd ASE curated proof of concepts, comments to the researcher or Bugcrowd (public or private), assignees, attachments, and state change metadata is available in the Crowdcontrol Platform.

Type of Vulnerability

IDOR

Additional Information to Properly describe impact

1. License Plate contains 8 characters (2 letters + 4 numbers + 2 letters).
2. 2 letters - region code of Ukraine.
3. 4 numbers - random numeric.
4. 2 letters - serial number.

So, it's not too hard to enumerate all license plates in DB.

Steps to Reproduce

1. Download script `exploit-idor.py` (this script use python 3).
2. Replace in this script authentication data (`user_id` and `token`)
3. Install package `requests` (https://pypi.org/project/requests/): `pip3 install requests` (if not installed)
4. Run this script with `python3 exploit-idor.py`.
5. When data will be loaded you will see output with all license plates from demo-server (in this case data is almost same).

PoC (first 6 users data)

```
KA7001AX Data:
[{'docStatus': 200, 'licensePlate': 'KA7001AX', 'vin': '1', 'vehicleLicen
seId': '1', 'serialNumber': '140457001', 'status': 'Active', 'expirationD
ate': '11.12.2020', 'name': 'XXXX', 'address': 'XXXX', 'phone': 'XXXX', '
email': 'xxxx', 'website': 'xxxx', 'validOn': '11.12.2020'}]
KA7002AX Data:
[{'docStatus': 200, 'licensePlate': 'KA7002AX', 'vin': '1', 'vehicleLicen
seId': '1', 'serialNumber': '140457002', 'status': 'Active', 'expirationD
ate': '11.12.2020', 'name': 'XXXX', 'address': 'XXXX', 'phone': 'XXXX', '
email': 'xxxx', 'website': 'xxxx', 'validOn': '11.12.2020'}]
```

```
KA7003AX Data:
[{'docStatus': 200, 'licensePlate': 'KA7003AX', 'vin': '1', 'vehicleLicen
seId': '1', 'serialNumber': '140457003', 'status': 'Active', 'expirationD
ate': '11.12.2020', 'name': 'XXXX', 'address': 'XXXX', 'phone': 'XXXX', '
email': 'xxxx', 'website': 'xxxx', 'validOn': '11.12.2020'}]
KA7004AX Data:
[{'docStatus': 200, 'licensePlate': 'KA7004AX', 'vin': '1', 'vehicleLicen
seId': '1', 'serialNumber': '140457004', 'status': 'Active', 'expirationD
ate': '11.12.2020', 'name': 'XXXX', 'address': 'XXXX', 'phone': 'XXXX', '
email': 'xxxx', 'website': 'xxxx', 'validOn': '11.12.2020'}]
KA7005AX Data:
[{'docStatus': 200, 'licensePlate': 'KA7005AX', 'vin': '1', 'vehicleLicen
seId': '1', 'serialNumber': '140457005', 'status': 'Active', 'expirationD
ate': '11.12.2020', 'name': 'XXXX', 'address': 'XXXX', 'phone': 'XXXX', '
email': 'xxxx', 'website': 'xxxx', 'validOn': '11.12.2020'}]
KA7006AX Data:
[{'docStatus': 200, 'licensePlate': 'KA7006AX', 'vin': '1', 'vehicleLicen
seId': '1', 'serialNumber': '140457006', 'status': 'Active', 'expirationD
ate': '11.12.2020', 'name': 'XXXX', 'address': 'XXXX', 'phone': 'XXXX', '
email': 'xxxx', 'website': 'xxxx', 'validOn': '11.12.2020'}]
```

Why

`vin` and `vehicleLicenseId` validated but not used for query. Only license plate is important, also, server don't validate access to this information.

Impact

Malicious user can steal sensitive information of other users like the following:`name`, `address`, `phone`, `email`, `website`, etc.

Mitigation

Validate if provided license plate belongs to current user.

Reference number:
c5e2b4db29198b1aeea4db550a9b11f025368dde99008f5621ffc4a7f44f502d

VRT:

Broken Access Control (BAC) > Insecure Direct Object References (IDOR)

Bug URL:

https://diia2sb.diia.gov.ua/api/v1/documents/vehicle-insurance

Extra info:

HTTP request:

App Crash on Malformed QR Code Read

P5

Type of Vulnerability

Improper Error Handling causing App Crash

Steps to Reproduce

1. Sign in to Diia app.
2. Open QR code reader.
3. Read QR code from section `QR code`
4. App crashed.

QR code

Embedded image redacted

Title: crashQR.png

Description

Part of Crash log

```
Thread 0 name:  Dispatch queue: com.apple.main-thread
Thread 0 Crashed:
0   Diia                          0x0000000104a8f234 0x1049c0000 + 8484
36
1   Diia                          0x0000000104a8f094 0x1049c0000 + 8480
20
2   Diia                          0x00000001049ebc08 0x1049c0000 + 1792
08
3   Diia                          0x0000000104b64d50 0x1049c0000 + 1723
728
4   Diia                          0x0000000104b66170 0x1049c0000 + 1728
880
5   Diia                          0x0000000104a7aaa8 0x1049c0000 + 7645
```

```
84
6   libdispatch.dylib               0x00000001b0155298 _dispatch_call_blo
ck_and_release + 24
7   libdispatch.dylib               0x00000001b0156280 _dispatch_client_c
allout + 16
8   libdispatch.dylib               0x00000001b01385b0 _dispatch_main_que
ue_callback_4CF$VARIANT$armv81 + 856
9   CoreFoundation                  0x00000001b049d5d0 __CFRUNLOOP_IS_SER
VICING_THE_MAIN_DISPATCH_QUEUE__ + 12
10  CoreFoundation                  0x00000001b0497a78 __CFRunLoopRun + 2
480
11  CoreFoundation                  0x00000001b0496b90 CFRunLoopRunSpecif
ic + 572
12  GraphicsServices                0x00000001c67b9598 GSEventRunModal +
160
13  UIKitCore                       0x00000001b2d80638 -[UIApplication _r
un] + 1052
14  UIKitCore                       0x00000001b2d85bb8 UIApplicationMain
+ 164
15  Diia                            0x00000001049c761c 0x1049c0000 + 3023
6
16  libdyld.dylib                   0x00000001b0175588 start + 4
```

Pseudo-C recovered by Ghidra on offset `0x100000000 + 848436 (0x1000cf234)`:

```
local_70 = 0x2f; // Slash
uStack104 = 0xe100000000000000;
local_60 = param_1;
lStack88 = param_2;
uVar3 = FUN_10000c2c8();
lVar4 = __stubs::_$sSy10FoundationE10components11separatedBySaySSGqd___tS
yRd__lF
                  (&local_70,__got::_$sSSN,__got::_$sSSN,uVar3,uVar3); //
 This is Array of substrings separated by "/".
// ...
  if (*(ulong *)(lVar4 + 0x10) == 5) { // If count equals to 5.
           /* WARNING: Could not recover jumptable at 0x0001000cf234. To
o many branches */
```

```
                /* WARNING: Treating indirect jump as call */
      UNRECOVERED_JUMPTABLE_00 = (code *)SoftwareBreakpoint(1,0x1000cf238);
    // ???
      (*UNRECOVERED_JUMPTABLE_00)();  // ???
      return;
}
```

Description:

Looks like parse of QR code crashed on invalid handling of path components.

Crash occurs, if parser can not find required string identifiers in path and count of path components is incorrect.

Additional information:

1. `components(separatedBy:)` -- this function called to split path by `0x2f` (`/`)
2. `lVar4` = `NSArray` instance.
3. `lVar4 + 0x10` = `NSArray.count` property (possibly).

Impact

Denial of Service.

App crashes and malicious user can use this for own purposes (as example, for social engineering).

Reference number:

10036f47783ba292943efd5eb0f63577e6a99694123e27da88ab2d7cf8ffd0d1

VRT:

Application-Level Denial-of-Service (DoS) > App Crash

Bug URL:

Extra info:

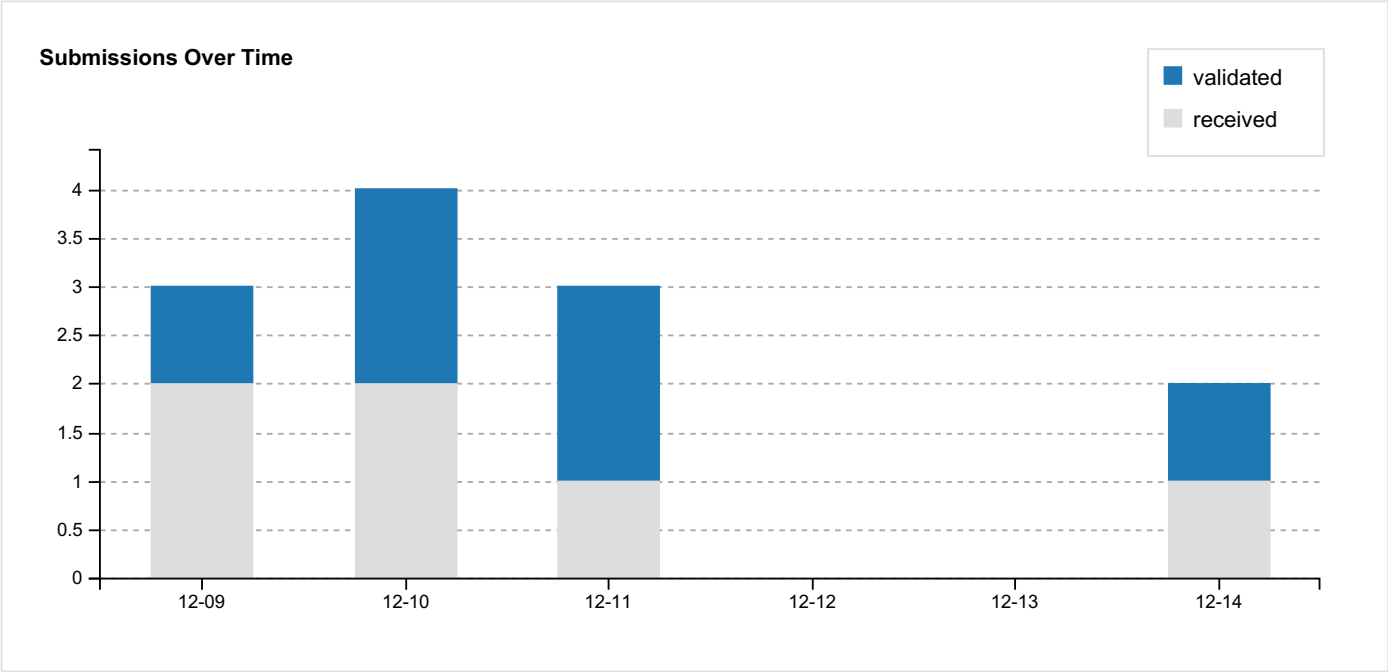
I attach video with reproduce.

HTTP request:

Included in this appendix are auxiliary metrics and insights into the On-Demand program. This includes information regarding submissions over time, payouts, prevalent issue types, and how the program compared to an aggregate of all Bugcrowd On-Demand programs.

Submissions over time

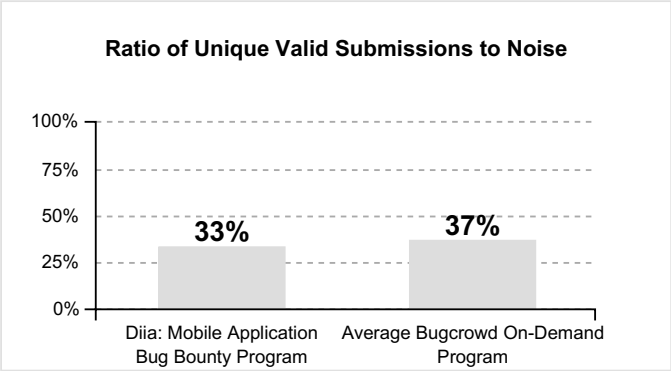
The timeline below shows submissions received and validated by the Bugcrowd team:



Submissions signal

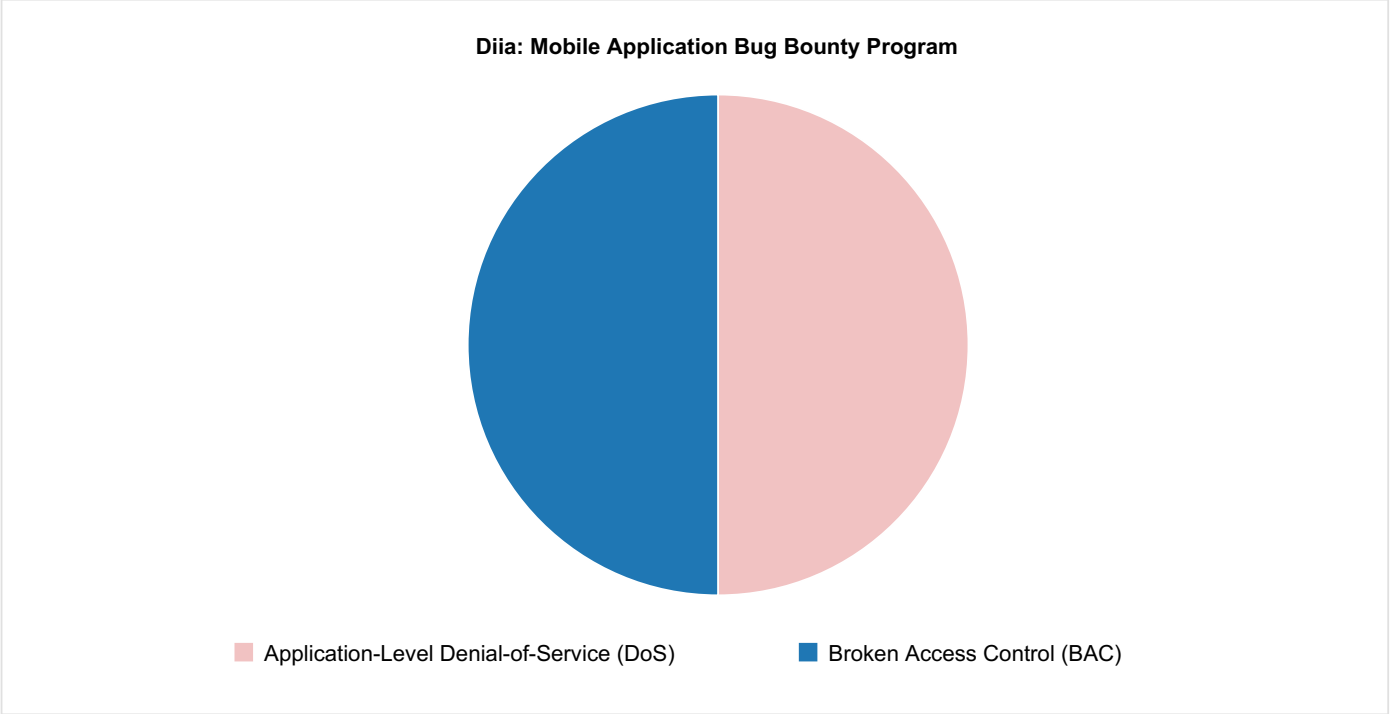
A total of **6** submissions were received, with **2** unique valid issues discovered. Bugcrowd identified **0** duplicate submissions and removed **4** invalid submissions. The ratio of unique valid submissions to noise was **33%**, which is lower than the average ratio of **37%** across Bugcrowd's other On-Demand programs.

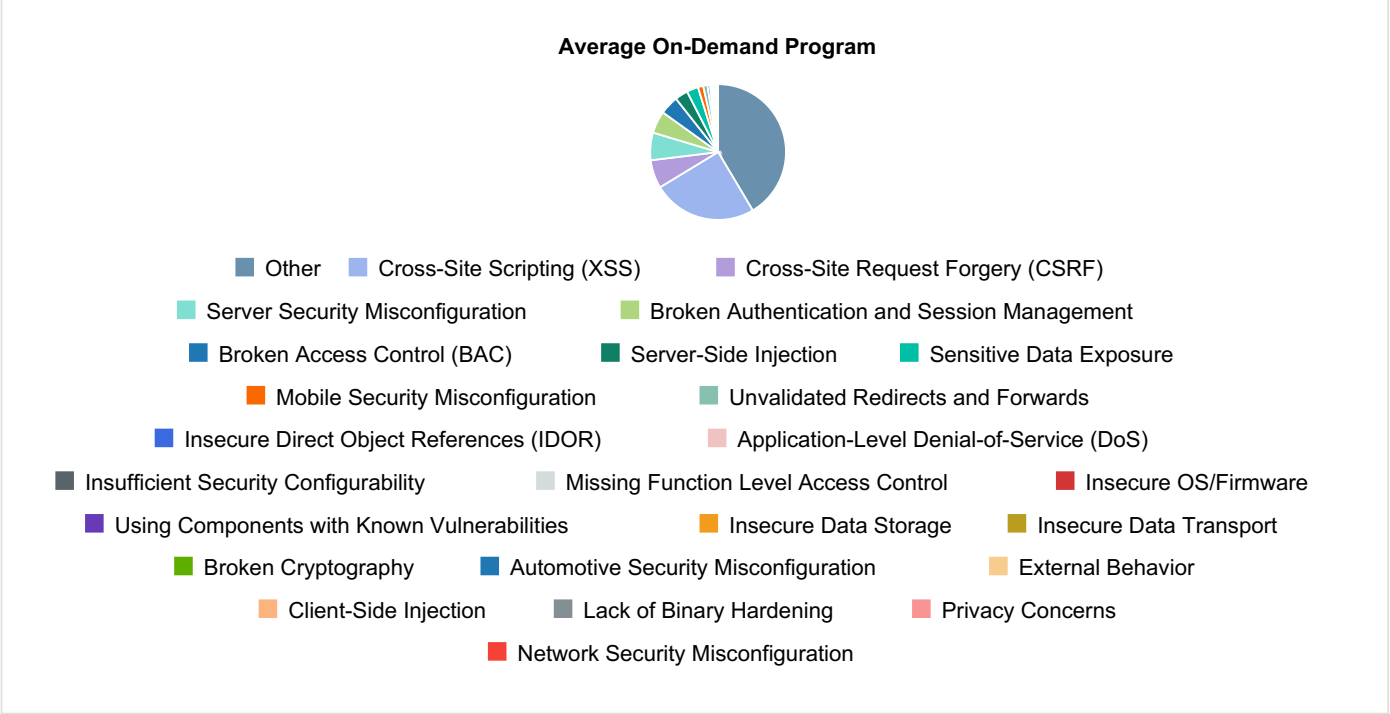
Submission Outcome	Count
Valid	2
Invalid	4
Duplicate	0
Total	**6**


Ratio of Unique Valid Submissions to Noise
100%
75%
50%
25%
0%
33%
37%
Diia: Mobile Application Bug Bounty Program
Average Bugcrowd On-Demand Program

Bug types overview

A comparison of the distribution of submissions across bug types for the On-Demand program to that of Bugcrowd's other On-Demand programs is shown below.





Closing statement

bugcrowd

December 17, 2020

Bugcrowd Inc.
921 Front St
Suite 100
San Francisco, CA 94111

Introduction

Between the dates of **12/08/2020 - 12/15/2020, Diia: Mobile Application Bug Bounty Program** engaged Bugcrowd Inc. to perform an On-Demand Bounty Program. During this time, **4** researchers from Bugcrowd submitted a total of **6** vulnerability submissions against **Diia: Mobile Application Bug Bounty Program's** targets. The purpose of this assessment was to identify security issues that could adversely affect the integrity of Diia: Mobile Application Bug Bounty Program. Testing focused on the following:

1. **DIIA Android Application**
2. **DIIA iOS Application**
3. **DIIA API**

The assessment was performed under the guidelines provided in the statement of work between **Diia: Mobile Application Bug Bounty Program** and Bugcrowd. This letter provides a high-level overview of the testing performed, and the result of that testing.

On-Demand Program Overview

An On-Demand Program is a novel approach to a penetration test. Traditional penetration tests use only one or two researchers to test an entire scope of work, while an On-Demand Program leverages a crowd of security researchers. This increases the probability of discovering esoteric issues that automated testing cannot find and that traditional vulnerability assessments may miss, in the same testing period.

It is important to note that this document represents a point-in-time evaluation of security posture. Security threats and attacker techniques evolve rapidly, and the results of this assessment are not intended to represent an endorsement of the adequacy of current security measures against future threats. This document contains information in summary form and is therefore intended for general guidance only; it is not intended as a substitute for detailed research or the exercise of professional judgment. The information presented here should not be construed as professional advice or service.

Testing Methods

This security assessment leveraged researchers that used a combination of proprietary, public, automated, and manual test techniques throughout the assessment. Commonly tested vulnerabilities include code injection, cross-site request forgery, cross-site scripting, insecure storage of sensitive data, authorization/authentication vulnerabilities, business logic vulnerabilities, and more.

Summary of Findings

During the program, Bugcrowd discovered the following:

Count	Technical Severity
0	Critical vulnerabilities
0	High vulnerabilities
0	Medium vulnerabilities
1	Low vulnerability
0	Informational findings

Upon completion of the assessment, all findings were reported to **Diia: Mobile Application Bug Bounty Program** along with all associated vulnerability data.